Министерство образования и науки Российской Федерации
ФГБОУ ВПО «Удмуртский государственный университет»

А. М. Сивков

ПРОГРАММИРОВАНИЕ

Руководство по выполнению лабораторных работ

Ижевск
2011

УДК 681.3
ББК 32.973
С 54

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент: к.ф.-м.н., доцент **М.А. Клочков**

С54 **Сивков А.М.**
Программирование. Руководство по выполнению лабораторных работ. Ижевск: Изд-во «Удмуртский университет», 2011. - 89 с.

Пособие содержит общие сведения по основам программирования на алгоритмическом языке высокого уровня, описание лабораторных работ и индивидуальные задания для самостоятельной работы. Предназначено для студентов бакалавриата, приступающих к изучению базовых средств языков программирования.

УДК 681.3
ББК 32.973

СОДЕРЖАНИЕ

ЛАБОРАТОРНЫЕ РАБОТЫ

ЗАДАЧИ ПО ПРОГРАММИРОВАНИЮ

ПРИЛОЖЕНИЕ

ПРЕДИСЛОВИЕ

Для обучения началам программирования обычно используют удобные для новичков алгоритмические языки Basic и Pascal. Однако, для написания серьёзных приложений эти языки применяются редко. Стандартом де-факто для профессионалов являются такие сложные и не всегда удобные языки как C++ и Java, и в дальнейшем изучающим программирование приходится переучиваться. При этом они делают в компьютерных программах специфические ошибки, связанные с использованием прежних усвоенных стереотипов в новых обстоятельствах, когда это уже неуместно.

Очевидно, разумнее было бы с самого начала подготовки студентов применять не учебный, а «боевой» язык программирования, но при этом, чтобы разом не отбить охоту к профессии, на первых порах использовать лишь простейшие базовые средства языка, и только после полного их освоения переходить к более сложным вещам. К сожалению, авторам большинства учебников по языку C++ не удаётся последовательно сохранить простоту изложения.

Предлагаемое пособие призвано исправить это положение. Оно рассчитано на читателя, который совсем не знаком с теорией и практикой программирования, и содержит всё необходимое для приобретения первоначальных теоретических и практических знаний. Для освоения материала требуется лишь общая компьютерная грамотность, то есть умение обращаться с папками и файлами, набирать тексты и т.п. Это пособие может быть использовано как самоучитель. При этом читать его следует за компьютером, практически выполняя все упражнения и обдумывая полученные результаты.

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1. Алгоритмы

В этой лабораторной работе не требуется составлять какие-либо программы для компьютера. Все действия, указанные в алгоритмах, нужно выполнить «вручную».

Общие сведения

1. Понятие алгоритма

Разработка компьютерной программы всегда начинается с осмысления задачи, которую с помощью компьютера необходимо решить, и построения подробного алгоритма.

Алгоритм - описание последовательности действий, приводящих к достижению цели, или, иными словами, к решению задачи.

Вообще говоря, в зависимости от задачи, алгоритм может иметь различный вид:

1. Словесные инструкции.
   Пример: выжми педаль сцепления;
   поверни ключ зажигания до упора и удерживай его в этом положении;
   когда двигатель заработает, отпусти ключ зажигания;
   медленно отпусти педаль сцепления.
2. Математическая формула.
   Пример: $y = x^2 + 1$
   То есть: умножь число, обозначенное буквой x, само на себя; прибавь к результату умножения единицу;
   полученное число обозначь буквой y.
3. Компьютерная программа.
   Пример:
```
while ( k < 3 )
{
   cout << k;
   k = k + 1;
}
```
   То есть: пока значение переменной k меньше трёх, повторяй следующие действия:
   1) выведи на экран значение переменной k;
   2) к значению переменной k прибавь единицу, и результат присвой переменной k.

Возможны и другие формы записи алгоритмов.

При разработке конкретного алгоритма необходимо учитывать возможности исполнителя, для которого предназначен алгоритм. Не стоит объяснять домашней собаке, как ей отвечать на телефонные звонки, когда Вас нет дома: она всё равно не сможет это сделать. Но её можно научить приносить Вам тапочки. Соответственно, если алгоритм предназначен для компьютера, все описываемые в нем действия должны сводиться к простейшим арифметическим операциям с числами, чтению из памяти и записи в память.

Компьютер выгодно применять для решения следующих задач:

- задачи, где количество исходных данных очень мало, но требуются весьма сложные вычисления;
- задачи, где количество исходных данных очень велико, но обработка их является простой;
- задачи с очень большим количеством исходных данных и очень сложной их обработкой.

Далее будут рассмотрены примеры некоторых типичных простых алгоритмов решения задач первого и второго типа.

2. Нахождение косинуса заданного угла

Математические вычисления всегда производятся по алгоритмам, иногда на удивление сложным. Но здесь разберем сравнительно простой пример. Пусть задача состоит в том, что нужно найти косинус какого-нибудь заданного в градусах угла. Будем, как принято в математике, «танцевать от печки», то есть рассмотрим определение.

По определению, косинус - это число, равное отношению длины катета к длине гипотенузы некоторого прямоугольного треугольника. Согласно этому, алгоритм нахождения косинуса угла α должен быть таким:

1. С помощью транспортира и линейки построить прямоугольный треугольник с углом α при вершине.
2. Измерить длину прилежащего катета и длину гипотенузы.
3. Разделить длину катета на длину гипотенузы.

К сожалению, в этом алгоритме не все хорошо. Даже если этот «танец» с транспортиром и линейкой исполняется человеком, возникает трудность точного построения заданного угла. Длину сторон треугольника можно измерить тоже только приближенно.

Ну а для компьютера такой алгоритм и вовсе не годится, так как процессор компьютера для геометрических построений и измерений не предназначен.

К счастью, такой уж неизбежной необходимости заниматься построением треугольника - нет. Нахождение косинуса угла можно свести к вычислению значения полинома (алгебраического многочлена) определенного вида. Это очень хорошо потому, что при вычислении значения полинома применяются только арифметические операции, и для такой деятельности компьютер идеально подходит.

Для случая, когда x – угол, заданный в радианах, полином, «представляющий» косинус, имеет вид:

$$\cos(x) = 1 - \frac{x^2}{1 \cdot 2} + \frac{x^4}{1 \cdot 2 \cdot 3 \cdot 4} - \frac{x^6}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} + \dots \quad (1)$$

Внимательно изучая формулу (1), можно кое-что заметить. Во-первых, «волшебный» полином содержит только чётные степени x. Во-вторых, в знаменателе каждого слагаемого стоит произведение последовательности натуральных чисел, начиная от единицы и кончая числом, равным показателю степени x. В-третьих, знаки слагаемых - плюс или минус - чередуются. Многоточие в конце формулы означает, что этот полином, вообще говоря, имеет бесконечное число членов. В высшей математике такой полином принято называть степенным рядом. С ростом номера, члены этого ряда очень быстро убывают, так что при вычислениях суммировать бесконечное число слагаемых не потребуется. Просто нужно иметь в виду, что от количества учтённых членов зависит, насколько результат расчёта будет отличаться от истинного значения косинуса.

Для сравнения, ниже приведено истинное значение косинуса угла π/3 и вычисленные значения полинома с двумя, тремя и четырьмя членами.

Истинное значение cos (π/3) 0,5000

Значение полинома с 2-мя членами 0,4517
Значение полинома с 3-мя членами 0,5018
Значение полинома с 4-мя членами 0,4999

Из приведённой таблицы видно, что чем больше учтено членов полинома, тем его значение ближе к истинному значению косинуса.

На основе слегка преобразованной формулы (1), можно предложить такой «экономный» компьютерный алгоритм нахождения cos (α), если угол α задан в градусах:

1. Преобразовать значение угла из градусов в радианы.
$$x = \alpha \cdot \pi \frac{}{} 180$$
2. Вычислить квадрат аргумента
$$y = x^2$$
3. Вычислить второй член полинома
$$p = \frac{y}{1 \cdot 2}$$
4. Вычислить третий член полинома
$$q = p \cdot \frac{y}{3 \cdot 4}$$
5. Вычислить четвертый член полинома
$$r = q \cdot \frac{y}{5 \cdot 6}$$
6. Сложить все члены с учетом знака
$$\cos(\alpha) = 1 - p + q - r$$

По сравнению с «простодушным» вычислением по формуле (1), этот алгоритм содержит меньшее количество арифметических действий. Исполнять его несложно: все арифметические действия выполняются по порядку от начала до конца.

3. Извлечение корня квадратного

Рассмотрим еще один пример вычислительного алгоритма: извлечение корня квадратного из заданного числа.

$$y = \sqrt{x}$$

По определению, корень квадратный из x - это такое число y, что если его возвести в квадрат (умножить само на себя) - получится x .

$$y^2 = x$$

Следовательно, отыскание корня квадратного, в принципе, можно поручить исполнителю, умеющему выполнять только две арифметические операции - умножение чисел и сравнение чисел.

Пусть корень квадратный из x требуется найти с точностью до двух десятичных знаков после запятой. Для решения этой задачи подойдет, например, следующий алгоритм.

Алгоритм поиска корня квадратного из x методом «перебора чисел»:

1. Составить список целых чисел от нуля до числа, следующего за x.
2. Последовательным перебором по списку найти два соседних целых числа, таких, что их квадраты соответственно меньше и больше, чем x. Первое из них принять за целую часть корня квадратного.

3. Составить список дробных чисел с одним десятичным знаком после запятой, начиная от меньшего и кончая большим из двух целых чисел, найденных в пункте 2.
4. Перебором по списку найти два соседних дробных числа с одним знаком после запятой, квадраты которых соответственно меньше и больше, чем x. Первое из них принять за приближенную дробную часть корня квадратного.
5. Аналогично найти второй знак после запятой.

Пример применения алгоритма для x=7:

1. Составить список целых чисел.

Список чисел	Умножение и сравнение
0	$0^2 = 0$ (<7)
1	$1^2 = 1$ (<7)
2	$2^2 = 4$ (<7)
3	$3^2 = 9$ (>7)
4	поиск закончен
5	
6	
7	
8	

Пара соседних целых чисел, одно из которых меньше искомого корня, а другое больше его, найдена. Это 2 и 3. Вывод: целая часть корня из 7 равна 2.

2. Составить список дробных чисел с одним знаком после запятой в интервале от 2,0 до 3,0.

Список чисел	Умножение и сравнение
2,0	$2,0^2 = 4,00$ (<7)
2,1	$2,1^2 = 4,41$ (<7)
2,2	$2,2^2 = 4,84$ (<7)
2,3	$2,3^2 = 5,29$ (<7)
2,4	$2,4^2 = 5,76$ (<7)
2,5	$2,5^2 = 6,25$ (<7)
2,6	$2,6^2 = 6,76$ (<7)
2,7	$2,7^2 = 7,29$ (>7)
2,8	поиск закончен
2,9	
3,0	

Пара соседних дробных чисел найдена. Это 2,6 и 2,7. Вывод: приближенное значение корня из 7 равно 2,6.

3. Составить список дробных чисел с двумя знаками после запятой в интервале от 2,60 до 2,70.

Список чисел	Умножение и сравнение
2,60	$2{,}60^2 = 6{,}7600$ (<7)
2,61	$2{,}61^2 = 6{,}8121$ (<7)
2,62	$2{,}62^2 = 6{,}8644$ (<7)
2,63	$2{,}63^2 = 6{,}9169$ (<7)
2,64	$2{,}64^2 = 6{,}9696$ (<7)
2,65	$2{,}65^2 = 7{,}0225$ (>7)
2,66	поиск закончен
2,67	
2,68	
2,69	
2,70	

Пара соседних дробных чисел найдена. Это 2,64 и 2,65. Вывод: приближенное значение корня из 7 - равно 2,64. (В действительности, в данном случае, 2,65 – более близкая оценка.)

Этот алгоритм по своему характеру сильно отличается от алгоритма, приведённого в предыдущем параграфе. Здесь при движении по списку чисел многократно повторяются одни и те же действия с числами. Прекращение или продолжение движения по списку зависит от выполнения некоторого условия, которое необходимо на каждом шаге проверять.

4. Сортировка списка

Список – это хранимая на бумаге или в памяти компьютера последовательность объектов. Например, список фамилий владельцев огнестрельного оружия, с указанием даты приобретения оружия:

25.09.2008 Сидоров
07.03.2010 Петров
31.12.2010 Иванов
12.04.2011 Фролов
...

Подобный список естественным путем возникает, если работники оружейного магазина добросовестно регистрируют покупки в специальном журнале. Если пофантазировать на тему всевозможных происшествий, то нетрудно сообразить, что у полиции иногда может возникнуть необходимость поискать в этом списке нужную запись.

Пока список не очень велик, задача поиска нужной записи решается просмотром всего списка человеком. Если же список содержит большое количество записей (например, сотни тысяч или миллионы), задача становится слишком трудоемкой, и ее лучше поручить компьютеру.

Несмотря на свое название (английское слово «компьютер» переводится как «вычислитель»), большинство компьютеров в мире занимаются не столько вычислениями, сколько именно работой со списками.

Трудоемкость поиска нужной записи в списке, конечно, зависит от того, упорядочен ли список по какому-нибудь признаку. В приведенном выше примере список фамилий упорядочен по дате приобретения оружия. Поэтому фамилию покупателя, который совершил покупку в известный день, найти легко.

Если найти нужную запись требуется не по дате покупки, а по фамилии покупателя, то и упорядочение списка лучше произвести по фамилии, по алфавиту.

Для упорядочения списка по алфавиту необходимо условиться о том, как сравнивать две произвольные фамилии, то есть ввести правило для использования знаков > («больше») и < («меньше») применительно к буквам. Это нетрудно, если буквы алфавита пронумерованы, то есть имеют числовые коды. При сравнении фамилий сравниваются коды их первой буквы. Если первая буква двух фамилий одинакова, сравниваются вторые буквы, если и они одинаковы – третьи и т.д. Согласно этому правилу можно считать, что Петров «больше», чем Иванов, а Сидоров «меньше», чем Фролов (даже если высоченный Сидоров и невысокий Фролов думают иначе).

Чтобы проверить, упорядочен ли список фамилий, нужно просмотреть его весь и убедиться, что каждый следующий элемент списка «больше» предыдущего элемента. При этом список можно изображать не только столбцом, но и строкой.

Например, список

Сидоров Петров Иванов Фролов

- неупорядочен,

а список:

Иванов Петров Сидоров Фролов

– упорядочен.

Упорядочение первоначально неупорядоченного списка иногда называют сортировкой. Рассмотрим, для примера, один из алгоритмов сортировки - не самый быстрый, но зато очень простой. Этот алгоритм получил название «пузырьковый», так как в нем «легкие» элементы списка по мере сортировки «всплывают» вверх, как пузырьки углекислого газа в стаканчике с газировкой. (У Вас, возможно, появилось желание немедленно сбегать за газировкой, но прежде давайте изучим алгоритм.)

«Пузырьковый» алгоритм сортировки:

1. Проверить, упорядочен ли список. Если список упорядочен - закончить работу, если нет – перейти к пункту 2.
2. Перейти к первому элементу списка.
3. Если следующий элемент списка меньше выбранного, поменять их местами.
4. Перейти к следующему элементу списка. Если он последний, перейти к пункту 1, если нет - перейти к пункту 3.

Пример применения алгоритма:

Пункт 1: Проверить, упорядочен ли список.

Сидоров
Петров
Иванов
Фролов

Нет, список не упорядочен. Нужно перейти к пункту 2.

Пункт 2: Перейти к первому элементу списка (установить на него указатель).

Сидоров <= (указатель)
Петров
Иванов
Фролов

Пункт 3: Если следующий элемент списка меньше выбранного, поменять их местами.

а)

. *Петров* <=
Сидоров
Иванов
Фролов

Освободили место в списке, выдвинув из него заменяемый элемент.

б)

Сидоров *Петров* <=
.
Иванов
Фролов

Переместили заменяющий элемент на освобожденное место.

в)

Сидоров <= (Указатель на прежнем месте.)
Петров
Иванов
Фролов

Обмен элементов произведен.

Пункт 4: Перейти к следующему элементу.

Петров
Сидоров <=
Иванов
Фролов

Опять пункт 3: Если следующий элемент списка меньше выбранного, поменять их местами.

Петров
Сидоров <=
Иванов
Фролов

Да, меньше. Меняем местами.

Петров
Иванов <=
Сидоров
Фролов

Обмен элементов произведен.

и так далее до конца первого прохода списка.

Список станет полностью упорядоченным, когда на втором проходе списка Иванов и Петров поменяются местами.

Это типичная задача и типичный алгоритм с большим количеством исходных данных и простыми действиями по их обработке. В подобных алгоритмах всегда используется цикличность действий и проверка выполнения некоторых условий.

Задания к лабораторной работе:

1. Найти косинус указанного угла.

Угол, заданный в градусах, взять из приведенной ниже таблицы, соответственно своему номеру варианта

вариант	угол	вариант	угол	вариант	угол	вариант	угол	вариант	угол
1	32	2	33	3	34	4	35	5	36
6	37	7	38	8	39	9	40	10	41
11	42	12	43	13	44	14	45	15	46
16	47	17	48	18	49	19	50	20	51
21	52	22	53	23	54	24	55	25	56

Косинус требуется найти двумя способами:

- с помощью графических построений и измерений (для увеличения точности измерений размеры треугольника должны быть достаточно большими);
- с помощью вычислений по формуле (1) разложения косинуса в степенной ряд (в формуле и в алгоритме учесть пять членов: вид пятого члена установить самостоятельно, по аналогии с предыдущими членами).

В отчете представить:

- геометрические построения, результаты измерений и величину косинуса, найденную делением длин отрезков (результат деления привести с тремя знаками после запятой);
- результаты вычислений вспомогательных величин и каждого слагаемого формулы (1) с количеством значащих цифр после запятой не менее девяти;
- вычисленное с помощью степенного ряда значение косинуса;
- значение косинуса, найденное с помощью стандартной функции встроенного инженерного калькулятора Windows (принять за эталон).
- определить количество верных знаков после запятой для значений косинуса, найденных с помощью графического и численного алгоритмов.

2. Извлечь корень квадратный из заданного числа с точностью четыре знака после запятой.

Заданное число взять из таблицы, соответственно номеру своего варианта.

вариант	число	вариант	Число	вариант	Число	вариант	число	вариант	число
1	27	2	41	3	30	4	15	5	13
6	8	7	47	8	31	9	20	10	11
11	29	12	55	13	51	14	44	15	40
16	5	17	23	18	22	19	39	20	53
21	42	22	35	23	46	24	24	25	19

В отчете привести протокол работы с указанием результатов вычислений на каждом шаге.

3. Выполнить сортировку списка по «пузырьковому» алгоритму.

Список, представляющий собой последовательность из семи букв, взять из приведённой ниже таблицы, соответственно номеру своего варианта.

№ варианта	Список букв
1	А Р Б А Л Е Т
2	Б Е Н Е Ф И С
3	В И Н О К У Р
4	Г А Л О Г Е Н
5	Д Ы Р О К О Л
6	Е В Д О К И М
7	Ж У Р А В Л Ь
8	З А Б Р А Л О
9	И З М О Р О М
10	К А Р А К У М
11	Л О Г О П Е Д
12	М А Р О Д Ё Р
13	Н О С О Р О Г
14	О Б О Б Р А Л
15	П И К А Д О Р
16	Р У Д О К О П
17	С А М О К А Т
18	Т А Р А К А Н
19	У Д И В И Л А
20	Ф И Л О С О Ф
21	Х А Р И Т О Н
22	Ц В Е Т Н И К
23	Ч Е М О Д А Н
24	Ш А П О Ч К А
25	Э Н Ш Т Е Й Н

Для наглядности, при сортировке можно пользоваться бумажными карточками с буквами и карточкой со стрелкой-указателем на текущий элемент списка. В отчет включить подробный протокол сортировки, в котором на каждом шаге показать вид списка и положение указателя. Подсчитать и привести в отчёте количество полных проходов списка до окончания сортировки.

Лабораторная работа № 2.
Технология создания программы

В этой лабораторной работе требуется освоить основы технологии создания программы, начиная от подготовки исходного текста и заканчивая получением готовой к исполнению машинной программы.

Общие сведения

1. Технология создания программы.

Чтобы создать готовую для использования компьютерную программу (неважно, простую или сложную) необходимы следующие действия:

- подготовка исходного текста программы
- компиляция исходного текста
- компоновка программы

Исходный текст – это запись подробного алгоритма решения задачи на особом языке, удобном для восприятия человеком. Его называют языком программирования высокого уровня. Примеры таких языков: Basic, Pascal, C++, Java. Подготовка исходного текста программы – «ручная» работа, она выполняется программистом с помощью текстового редактора.

Программа, записанная на языке высокого уровня, понятна человеку, но не может быть непосредственно исполнена процессором компьютера. Чтобы процессор мог воспринимать программу и выполнять ее, программа предварительно должна быть переведена с языка высокого уровня на язык низкого уровня – машинный язык. Перевод исходного текста программы на машинный язык называют компиляцией программы. Такой перевод может быть выполнен автоматически, с помощью специальной компьютерной программы – компилятора.

Однако в результате компиляции еще не получается готовая к использованию программа, а получается всего лишь «полуфабрикат». Его необходимо объединить со специальными подпрограммами, которые «умеют» выполнять часто встречающиеся стандартные действия, например, вывод символов на экран, чтение символов, набираемых на клавиатуре, вычисление математических функций и тому подобное. Подпрограммы, предназначенные для выполнения таких стандартных действий, подготовлены другими программистами и помещены в так называемые библиотеки подпрограмм. Объединение вновь создаваемой программы с библиотечными подпрограммами называется компоновкой и выполняется автоматически с помощью специальной программы-компоновщика.

Для удобства программиста текстовый редактор, компилятор и компоновщик часто собраны в единую среду разработки программ и легко запускаются нажатием определенного сочетания клавиш или с помощью выбора мышью пунктов меню. Благодаря этому, на практике создание небольшой компьютерной программы не вызывает трудностей даже у новичка.

Отличным примером среды разработки является Microsoft Visual C++ Express Edition для операционной системы Windows. Эту среду можно бесплатно получить на официальном сайте Microsoft. Простейшие приемы работы с ней описаны в Приложении.

Прежде, чем выполнять упражнения, описанные в следующих параграфах, обязательно прочтите

Приложение к этому пособию и практически проделайте все указанные там действия.

2. Первая программа. Оператор вывода.

Запустите среду программирования Visual C++ и в окне текстового редактора наберите очень простую программу, предназначенную для сложения чисел 5 и 7:

```
#include <iostream>
using namespace std;
int main()
{
        cout << 5+7;
}
```

Выполните компиляцию программы (в Visual C++ для этого можно просто нажать клавишу F7). Убедитесь, что компиляция и компоновка программы прошли успешно и сообщения об ошибках отсутствуют.

Если, набирая программу, Вы допустили ошибку, компилятор не сможет успешно перевести программу на машинный язык. Тогда он выдаст Вам соответствующее сообщение. В этом случае, внимательно, буква за буквой, проверьте набранный Вами исходный текст, найдите ошибку и исправьте её.

Если компиляция и компоновка прошли успешно, запустите программу на выполнение (в Visual C++ для этого нужно нажать сочетание клавиш Ctrl + F5). При этом появится окно консоли с текстом:

```
12Для продолжения нажмите любую клавишу. . .
```

Это означает, что Ваша программа работает! На экран выведен результат сложения (число 12) и приглашение нажать любую клавишу. Нажатие какой-нибудь клавиши приводит к закрытию окна консоли.

Теперь разберем, как устроена программа. Она очень проста и состоит из шести строк.

Единственно интересной для Вас является строка номер пять. Именно в этой строке содержатся команда сложения двух чисел (5+7) и команда вывода результата на экран (cout <<).

Остальные строки – вспомогательные, служебные. Первая и вторая строки необходимы компилятору, чтобы правильно перевести на машинный язык команду вывода. Третья строка является своего рода заголовком содержательной части программы. Фигурные скобки в четвертой и шестой строках обрамляют «тело» содержательной части программы.

3. Эксперименты с первой программой.

Если Вам удалось запустить первую программу, то Вы уже получили заряд оптимизма. Однако можно получить еще больше удовольствия и новых знаний, если проделать с ней несколько интересных экспериментов.

Вы, конечно, заметили, что программа выводит результат сложения на экран не очень изящно. Вплотную к числу 12, в той же строке, выводится еще и служебный текст. Чтобы вывод служебного текста произошёл иначе, добавьте в программу еще одну инструкцию:

```
#include <iostream>
using namespace std;
int main()
{
        cout << 5+7;
        cout << endl;
```

```
}
```

Не перепутайте латинскую букву l («эл») в слове endl с единицей: они очень похожи!

Откомпилируйте и запустите этот вариант программы. Обратите внимание, что на этот раз служебный текст появился на экране на одну строку ниже. Дело в том, что помещение в поток вывода слова endl - это указание перевести строку (endl - сокращение от англ. end of line – конец строки).

То же самое можно сделать и иначе, в одном предложении:

```
#include <iostream>
using namespace std;
int main()
{
        cout << 5+7 << endl;
}
```

Проверьте, как работает такая программа.

Теперь выполним эксперименты по «сокращению» программы. Уберите из программы операторы (команды), выполняющие сложение чисел и вывод результата на экран, и оставьте только такие строки:

```
#include <iostream>
using namespace std;
int main()
{
}
```

Откомпилируйте эту «пустую» программу и запустите её. Как видите, сообщений об ошибках нет. Всё работает. Значит, формально программа является правильной. Но она не выводит на экран ничего интересного, кроме приглашения нажать любую клавишу. Это и не удивительно, так как операторов, выполняющих какие-либо содержательные действия, в программе нет. Её можно рассматривать как минимальную «заготовку», «шаблон» для написания любой нужной Вам программы. Своего рода форму для выпекания вкусной булочки.

```
#include <iostream>
using namespace std;

int main()
{
        //На этом месте помещается всё
        //необходимое для решения задачи
}
```

Полезно знать, что текст, начинающийся с двойной косой черты, и до конца строки, не воспринимается компилятором и никак не влияет на работу программы. Программисты используют это для вставки в программу кратких пояснений, которые называют комментариями.

Компилятор также игнорирует пустые строки в программе. В тех местах, где в предложениях должен или может ставиться пробел, можно поставить и два пробела и больше. Использование дополнительных пробелов и пустых строк безразлично компилятору, но часто улучшает «читаемость» программы человеком.

Задания к лабораторной работе:

1. **Набрать исходный текст, откомпилировать и запустить программу сложения двух чисел**

   В отчете представить:
   - исходный текст программы (подчеркнуть цветными карандашами разным цветом: служебные строки; оператор сложения; оператор вывода на экран);
   - результат работы программы - точно так, как он был выведен в окно консоли.

2. **Проделать эксперименты с первой программой.**

   Переделать программу так, чтобы:
   - программа не выполняла никаких действий, но была правильной – «пустая» программа;
   - в одной программе вычислялись и выводились на экран в разных строках сумма, разность, произведение и частное от деления двух чисел.

   В отчете представить исходный текст и результаты работы программ.

Лабораторная работа № 3.
Переменные

В этой лабораторной работе требуется усвоить понятие переменной, изучить оператор ввода, оператор присваивания, познакомиться с записью чисел и арифметических выражений.

Общие сведения

1. Как научить программу общаться.

Рассмотренная нами первая программа безукоризненно выполняла сложение чисел 5 и 7. Эти исходные данные были заложены Вами в программу ещё при ее создании. А если кому-то понадобится сложить другие числа? Конечно, Вы можете открыть исходный текст этой гениальной программы в текстовом редакторе, изменить числа, вновь откомпилировать её и запустить. Но большинство людей хотели бы пользоваться только полностью готовыми программами и не хотели бы вникать в тонкости их создания.

Гораздо лучше было бы устроить всё так, чтобы при изменении исходных данных, программу не приходилось бы изменять и заново компилировать. Хотелось бы, чтобы уже откомпилированная, полностью готовая программа после запуска могла ждать, когда пользователь введёт с клавиатуры исходные данные (любые нужные числа). А после ввода данных, могла бы хранить их и по мере надобности использовать.

Рассмотрим пример такой «универсальной» программы. В одном из старых мультфильмов была замечательная

песенка: «Поделись улыбкою своей, и она к тебе не раз ещё вернётся».

Вот программа, которая, узнав от Вас, сколько раз Вы ей улыбнулись, сообщает, сколько раз она улыбнулась Вам:

```
#include <iostream>
using namespace std;

int main()
{
        int n;
        cin >> n;
        cout << n*2 << endl;
}
```

Наберите и запустите эту программу. Вы увидите на экране пустое окно терминала с мигающим в нем курсором. Это означает, что компьютер ждёт ввода какого-нибудь числа с клавиатуры. Если Вы наберёте целое число (улыбок) на клавиатуре и нажмёте клавишу Enter, то на экран будет выведено ещё одно число, которое показывает, сколько раз компьютер «улыбнулся» Вам в ответ.

Обратите внимание, что после запуска этой программы Вы можете ввести с клавиатуры произвольное число. Каким бы оно ни было, программа будет работать. Вам не придётся что-либо менять в программе. В каком-то смысле программа универсальна по отношению к исходным данным. Как это достигнуто?

В этой программе для ввода и хранения исходных данных, то есть количества Ваших улыбок (конечно, это шутка), используется то, что программисты называют «переменная».

В самом простом случае компьютер можно представлять себе состоящим из четырёх частей:

- процессор - устройство для чтения и выполнения программы;
- оперативная память - устройство для временного хранения программы и обрабатываемых данных;
- монитор - устройство для отображения данных;
- клавиатура - устройство для ввода данных.

Переменная – это, попросту говоря, ячейка в оперативной памяти, вроде ячейки камеры хранения на вокзале или в супермаркете. Переменной она называется потому, что во время работы программы в разные моменты в этой ячейке может храниться то одно число, то другое.

Замечательно, что программисту не приходится думать о том, по какому именно адресу в оперативной памяти и сколько конкретно байтов нужно выделить для переменной. Это – забота компилятора. В программе нужно лишь описать переменную, то есть указать её тип, и её имя.

В приведённой выше программе описание переменной выглядит очень просто:

```
int n;
```

В этом описании: int - указание типа переменной (сокращение от англ. integer - целый); n - имя переменной. В сущности, это указание компилятору, что нужно выделить «где-нибудь в памяти» ячейку для хранения целых чисел и назначить этой ячейке имя n. (Имя переменной может состоять только из латинских букв, цифр и знака подчёркивания.)

Но выделить ячейку в памяти – это ещё не всё: нужно ещё каким-то образом заставить компьютер записать в неё желаемое число.

Строка программы

cin >> n;

представляет собой так называемый оператор ввода. Этот оператор, когда во время работы программы доходит очередь до его выполнения, производит чтение числа, введённого с клавиатуры, и запись этого числа в ячейку с указанным именем. У программистов это называется: «задать значение переменной».

В следующей строке программы вычисляется количество «улыбок компьютера» с помощью умножения переменной n на 2. Разумеется, компьютер умножает на 2 не саму ячейку в оперативной памяти, а хранящееся в этой ячейке число.

И наконец, результат умножения выводится на экран.

После того, как Вы убедитесь, что эта программа отлично работает, предлагаю проделать с ней небольшой эксперимент. После запуска программы введите с клавиатуры не целое число улыбок, а дробное. Например, введите число 2.5 (две целых, пять десятых). Дробную часть числа от целой в C++ принято отделять точкой.

Вы удивлены результатом? Не удивляйтесь. Вы ввели дробное число, а описанная в программе переменная может использоваться для ввода и хранения только целых чисел. Поэтому произошло отбрасывание дробной части числа. Как хранить и использовать дробные числа, будет рассказано в другом параграфе.

Рассмотренная сейчас программа, хотя и очень мала, но, на мой взгляд, чудесна. Ведь она может общаться с пользователем: «слушать» его и в ответ что-то «говорить» ему. Чуть-чуть похоже на искусственный интеллект. Жаль только, что общение этой программы с человеком - очень скупое. На экран выводятся только числа. Давайте научим программу выводить сообщения на привычном для нас языке.

Во-первых, хорошо было бы, если бы перед операцией ввода данных программа сообщила, чего же она будет ожидать от пользователя. Во-вторых, пользователю понравится, если программа пояснит, что означает число, которое она в итоге выведет на экран. Всё это можно сделать путём вывода в окно консоли поясняющего текста. Вывод текста на экран легко осуществить с помощью команды:

cout << "текст, который нужно вывести на зкран" << endl;

Между двойными кавычками можно записывать любой текст. Компилятор при переводе программы не вникает в его смысл. Текст будет выводиться на экран так, как он есть, буквально.

Вот более «разговорчивый» вариант программы:

```
#include <iostream>
using namespace std;

int main()
{
  int n;
  cout << "Vvedite kolichestvo Vashih ulibok" << endl;
  cin >> n;
  cout << "Kolichestvo ulibok kompjutera v otvet" << endl;
  cout << n*2 << endl;
}
```

Как Вы уже догадались, обычно команды выполняются в программе в той последовательности, в которой они записаны, одна за другой. С учётом этого, постарайтесь ещё до запуска этой программы предсказать, как она будет работать, и что Вы увидите на экране.

Вы удивлены, что русские слова записаны в программе латинскими буквами? Ну что ж, запишите их кириллицей и

посмотрите, что из этого получится. Немного посмеяться – полезно для здоровья.

4. Оператор присваивания. Запись чисел. Арифметические выражения.

Возможно, Вы считаете, что программирование – дело нешуточное, и писать программы нужно не для того, чтобы обмениваться с компьютером улыбочками. Ну что ж! Давайте тогда поставим более серьёзную задачу.

Представьте, что Вы мчитесь на автомобиле. Как определить, не слишком ли велика скорость? Ведь спидометр показывает километры в час. А если на дороге вдруг появится препятствие - счет уже пойдёт на метры и секунды.

Создайте программу, которая пересчитывает скорость из километров в час - в метры в секунду, а также вычисляет длину тормозного пути, то есть расстояние, необходимое для полной остановки автомобиля. (Кто знает - может быть, впоследствии, когда Вы усовершенствуете эту программу, фирма КАМАЗ купит её у Вас для своего нового гоночного автомобиля.)

Формула для пересчета скорости в метры в секунду очень проста: скорость в километрах в час нужно умножить на 1000 и разделить на 3600. (1 км = 1000 м, а 1 ч = 3600 с).

Вывод формулы для расчета тормозного пути требует уже более глубокого знания физики. Для расчета нужно будет задать величину коэффициента трения шин об асфальт и величину ускорения свободного падения.

Если Вы изрядно подзабыли физику, можете, конечно, не выводить никаких формул, а воспользоваться готовой программой:

```
//Программа для расчета скорости и тормозного пути

#include <iostream>
using namespace std;

int main()
{
//Описание переменных
  double    g, mju, v1, v2, s;
//Задание значений переменных
        g = 9.81;    //ускорение свободного падения в м/с^2
        mju = 0.7;  //коэффициент трения
        v1 = 36.0;  //скорость в км/ч
//Вывод на экран скорости в км/ч
        cout << "Skorost v kilometrah v chas" << endl;
        cout << v1 << endl;
//Вычисление скорости в м/с  и вывод её на экран
        v2 = v1*1000/3600;
        cout << "Skorost v metrah v sekundu" << endl;
        cout << v2 << endl;
//Вычисление тормозного пути в м и вывод его на экран
        s = v2*v2/(mju*g);
        cout << "Tormoznoj put v metrah" << endl;
        cout << s << endl;
}
```

Обсудим эту программу. Сразу после фигурной скобки в ней идёт описание переменных. Имена переменных даны списком, через запятую. Поскольку программа должна уметь работать и с целыми и с дробными числами (кто знает, какая у Вас может оказаться скорость?), для всех переменных указан тип double (сокращение от англ. double precision - двойная точность). Этот тип применяется для

дробных чисел. Впрочем, если к целому числу добавить нулевую дробную часть – оно тоже «прикинется» дробным числом и его можно будет хранить в переменной такого типа.

Выбор типа переменных является существенным для правильного решения задачи. Дробные числа невозможно разместить в переменных целого типа. Для их хранения требуется больше оперативной памяти, чем для хранения целых. Арифметические действия тоже производятся по-разному: с целыми числами – абсолютно точно, с дробными числами – вообще говоря, приближённо.

В приведённой программе для Вас есть и ещё кое-что новое. В этой программе переменные получают значения не с помощью оператора ввода, а с помощью оператора присваивания.

Оператор присваивания в общем случае имеет вид:

<имя переменной> = <арифметическое выражение>;

Знак " = " означает здесь вовсе не равенство двух величин друг другу, а нечто иное. Этот знак является указанием компьютеру присвоить переменной, указанной слева от знака " = ", значение, полученное в результате вычисления арифметического выражения, стоящего справа. Арифметическое выражение может принимать разный вид: это может быть формула, переменная или константа (явно записанное число).

Так, например, оператор

```
x = 2;
```

означает команду записать в ячейку с именем x число 2.

А выполнение оператора

```
y = (2.5 * x + 1)/2;
```

происходит в следующем порядке:

1. Константа 2.5 умножается на значение переменной x. (Если, например, переменной x ранее в программе было присвоено значение 2, то в результате получается 5.)
2. К результату первого действия прибавляется 1. (В результате получается 6.)
3. Результат второго действия делится на 2. (В результате получается 3.)
4. Окончательный результат вычисления выражения (число 3) присваивается переменной y.

Вернёмся к обсуждению нашей «автомобильной» программы.

Операторы присваивания

```
g = 9.81;
mju = 0.7;
v1 = 36.0;
```

задают:

- для ускорения свободного падения: g - значение 9,81 м/с^2,
- для коэффициента трения шин о дорожное покрытие: mju - значение 0,7 (очень зависит от типа и состояния дорожного покрытия),
- для скорости автомобиля: v1 - значение 36 км/ч (примерно с такой скоростью «мчались» автомобили в начале прошлого века).

В языке C++ числа (константы) можно записывать в разной форме.

Примеры записи чисел:

- целые: 7 -137

- д

  робные: -37.815 0.000125
- в

  экспоненциальной форме: -1.3e3 0.25e-2

Запись чисел в экспоненциальной форме означает запись с десятичным порядком, то есть, соответственно примеру $-1{,}3\cdot10^{3}$ (-1300), $0{,}25\cdot10^{-2}$ (0,0025). Это может быть удобным для записи очень больших или, наоборот, очень маленьких чисел.

Обратите внимание, что разделителем целой и дробной части числа в C++ является не запятая, а точка.

Ну и наконец, запустите приведённую выше программу и поэкспериментируйте с ней, меняя значение скорости и коэффициента трения. Может быть, проанализировав результаты расчётов, Вы начнёте иначе смотреть на любителей очень быстрой езды.

Задания к лабораторной работе:

1. **Набрать исходный текст, откомпилировать и запустить все программы, приведённые в описании работы**

   В отчете представить:
   - исходный текст программ (подчеркнуть цветными карандашами разным цветом строки: с описанием переменных, с оператором ввода, с оператором вывода, с оператором присваивания; выделить арифметические выражения);
   - результат работы программы, так, как он был выведен в окно консоли.

2. **Проделать эксперименты с последней программой.**

   - выполнить расчеты для пяти различных значений скорости;
   - построить два графика зависимости тормозного пути от скорости для двух разных значений коэффициента трения.

3. **Создать программу для вычисления кинетической энергии тела массой m, движущегося со скоростью v.**

   В отчете представить постановку задачи, формулы, исходный текст программы и результаты расчетов для нескольких значений массы и скорости тела.

Лабораторная работа № 4
Ветвление

В этой лабораторной работе требуется изучить условный оператор, познакомиться с примерами программ, выбирающих различные варианты действий в зависимости от ситуации.

Общие сведения

1. Условный оператор. Оператор перехода.

– Если выпадет «орел», пойдем на каток, если «решка» - в кино.
– А если монета на ребро встанет?
– Тогда пойдем в школу на урок.

Программы, которые мы до этого момента рассмотрели, работают весьма «прямолинейно», то есть, команды в этих программах выполняются одна за другой именно в той последовательности, в какой они записаны. Но в жизни бывают случаи, когда нужно проявить гибкость и отступить от «железного» порядка вычислений. Предположим, программа должна вычислять корень квадратный из числа, заданного пользователем, а пользователь, по недоразумению, ввел отрицательное число. В таком случае желательно было бы предусмотреть в программе проверку введённого числа, и если число отрицательное, не производить бессмысленные вычисления, а сразу выдать на экран предостерегающий текст.

Для проверки заданного условия, от которого должен зависеть выбор того или иного варианта действий в программе и переключения между двумя различными вариантами в языке C++ имеется так называемый условный оператор (оператор выбора):

if (<условие>) <оператор1>; else <оператор2>;

Условный оператор работает так:

- если (англ. “if” означает “если”) удовлетворяется указанное условие, будет выполнен «оператор1»;
- в противном случае (англ. “else” означает “иначе”) – «оператор2»;
- после того, как действия, предусмотренные в условном операторе, выполнены, управление передаётся следующему оператору программы.

Пример программы с использованием условного оператора:

```
//Программа с выбором варианта действий

#include <iostream>
using namespace std;

int main()
{
  int k;
  cout << "Vvedite chislo: orel - 1, reshka - 2" << endl;
  cin >> k;
  //Условный оператор
   if (k == 1)
         cout << "Idem na katok" << endl;     //оператор1
```

```
    else
        cout << "Idem v kino" << endl;        //оператор2
  //Продолжение программы
  cout << "Potom idem domoj" << endl;
}
```

Программа сообщает Вам, что нужно ввести с клавиатуры результат бросания монеты. Если выпал «орел», вводится 1, если «решка» - 2. Введённое число записывается в переменную k. Затем программа проверяет, равно ли k единице, и в зависимости от результата проверки сообщает Вам, идёте ли вы на каток или в кино. После этого управление предаётся оператору, следующему за условным оператором, и программа сообщает, куда вы идете после развлечений.

Если эта программа кажется Вам слишком несерьёзной, наберите другую, аналогичную программу:

```
//Программа вычисления корня квадратного

#include <iostream>
using namespace std;
//подключение библиотеки математических функций
#include <math.h>

int main()
{
  double x;
  cout << "Vvedite chislo" << endl;
  cin >> x;
//Выбор одного из двух вариантов действий
  if (x < 0)
    {
      //Вывод предостерегающего сообщения
        cout << "Nelzia vvodit otricatelnoe chislo.";
        cout << endl;
    }
   else
    {
     //Вывод результата вычисления корня
        cout << "Koren kvadratnij" << endl;
        cout << sqrt(x)<< endl;
    }
//Дальше - опять последов. выполнение программы
  cout << "Rabota programmy zakonchena" << endl;
}
```

Эта программа находит корень квадратный из введённого пользователем числа. Если пользователь ввёл отрицательное число, корень не вычисляется. Вместо этого на экран выводится предостерегающее сообщение.
Вычисление корня квадратного в программе производится с помощью использования функции sqrt() из библиотеки математических функций. Для того, чтобы можно было пользоваться этой библиотекой, в программу включена строка

```
#include <math.h>
```

В программе есть ещё одна особенность, на которую стоит обратить внимание. В условном операторе после if и после else по правилам можно ставить только одиночный оператор. Поскольку в данной программе в каждой ветви вычислений необходимо было поставить по два оператора, эти операторы пришлось заключить в фигурные скобки. Любая последовательность операторов, заключенная в фигурные скобки, формально рассматривается как

одиночный оператор, который принято называть «составным оператором».

Условие, заданное в операторе выбора, в программировании называется логическим выражением.

Примеры простых логических выражений:

x == 1 (значение x равно 1)

(a+b) > c (значение a+b больше, чем значение c)

Для построения логических выражений применяют знаки операций отношения:

== - равно, (два знака равенства, один за другим)

!= - не равно,

> – больше,

>= - больше или равно,

< – меньше,

<= - меньше или равно.

В результате вычисления значения логического выражения может получиться единица («истинно»), или ноль («ложно»).

Разрешается использовать и составные логические выражения. Так, например, двойное математическое неравенство 1 < x < 3 можно записать в виде составного логического выражения

(1 < x) && (x < 3)

В составных выражениях в C++ могут применяться следующие знаки логических операций:

&& - логическое «И»,

|| - логическое «ИЛИ»,

! - логическое «НЕ».

Условие, указанное в условном операторе, считается удовлетворенным, если соответствующее логическое выражение имеет значение «истинно».

Полезно знать, что условный оператор, кроме полной формы, имеет и сокращенную форму, без ветви else:

if (<условие>) <оператор1>;

В такой форме, если указанное в скобках условие удовлетворяется, будет выполнен «оператор1», в противном случае управление сразу перейдёт к следующему оператору программы.

```
//Программа, использующая сокращенную форму
условного оператора

#include <iostream>
using namespace std;

int main()
{
   int k;
   metka:   cout << "U popa bila sobaka - on ee ljubil...";
   cout << endl;
   cout << "Vvedite: povtorenie-1, dostatochno-2" << endl;
   cin >> k;
   if (k == 1) goto metka;   //возврат к началу
}
```

В этой программе, в качестве оператора, который нужно выполнить, если удовлетворяется заданное условие, использован оператор перехода goto (англ. «go to» означает «перейти к»). Этот оператор вызывает передачу управления другому оператору программы: тому, который помечен именем, указанным в операторе перехода. В данном случае использовано имя metka . После запуска программы Вы обнаружите, что выполнение операторов программы циклически повторяется, если на предложение программы ввести число - вводить единицу.

Хотя эта программа нормально работает, имейте в виду, что применять оператор перехода для организации повторений в настоящее время считается дурным вкусом. Такие программы трудно читать и изменять, если в этом вдруг возникла необходимость. Для организации повторений в языке C++ имеются специальные более удобные операторы.

Задания к лабораторной работе:

1. **Откомпилировать и запустить все программы, приведённые в описании этой работы**

   В отчете представить:
   - исходный текст программ (выделить цветными карандашами разным цветом ветви программы, выполняющиеся в зависимости от ситуации: красным цветом – ветвь, выполняющуюся, если условие, указанное в условном операторе, удовлетворяется; желтым цветом - если не удовлетворяется;
   - результат работы программы - как он был выведен в окно консоли - в случае выполнения операторов одной ветви и в случае выполнения операторов другой ветви.

2. **Составить программу, вычисляющую длину второго катета *b* прямоугольного треугольника, если с клавиатуры введены: длина его первого катета *a* и длина гипотенузы *c*.**

   В отчете представить текст программы и результаты вывода на экран для случая, когда гипотенуза длиннее первого катета и для случая, когда она - «короче».

Лабораторная работа № 5.
Циклы

В этой лабораторной работе требуется изучить операторы цикла for и while.

Общие сведения

1. Цикл. Оператор цикла “for”.

Одним из самых замечательных достоинств компьютера является то, что он способен автоматически повторять однообразные действия, очень быстро и без ошибок. Это позволяет с его помощью за малое время проделать работу, на которую у человека ушли бы годы. Именно поэтому, хотя составление программ требует времени и специальных знаний, игра стоит свеч.

Многократное исполнение одной и той же группы инструкций называется в программировании циклом (англ. circle – круг). Для организации циклов в C++ чаще всего используются два вида операторов: оператор “for” и оператор “while” . Оператор “for” (англ. for - для) имеет форму:

<заголовок цикла>
<тело цикла>

Тело цикла - это любой одиночный или составной оператор, выполнение которого требуется многократно повторять.

Заголовок цикла, в сущности, указывает, сколько раз должно быть выполнено тело цикла, и записывается чаще всего в виде:

for (<задание начального значения переменной цикла>; <условие, при удовлетворении которого следует выполнять тело цикла>; <изменение переменной цикла после каждого выполнения тела цикла>)

Переменная цикла – это, обычно, переменная целого типа, которая играет роль «счётчика» оборотов цикла. После каждого выполнения тела цикла её значение изменяется.

Кратко оператор “for” можно выразить словами: для указанных в скобках обстоятельств - повторяй выполнение тела цикла.

Чтобы посмотреть, как работает оператор цикла, наберите и запустите следующую простую программу, созданную «по мотивам» популярной детской считалочки: «Раз, два, три, четыре, пять! Вышел зайчик погулять!»:

```
//Программа-«считалочка»

#include <iostream>
using namespace std;

int main()
{
        int i;
        for (i=1; i<=5; i=i+1)      //заголовок цикла
            cout << i << endl;      //тело цикла
       //продолжение программы
        cout << "Vishel zajchik poguliat!" << endl;
}
```

В этой программе с помощью оператора цикла на экран последовательно выводятся числа от 1 до 5. После этого программа выводит на экран зайчика, «погулять».

Переменной-«счётчиком» оборотов цикла в приведённой программе служит переменная с именем i (в принципе, имя можно выбирать любое, но i - предмет неувядающей любви всех программистов). На каждом обороте цикла значение этой переменной последовательно увеличивается, и выводится на экран.

В заголовке цикла указано,

- какое начальное значение получает переменная-счётчик:
  for (i=1; ;) ,
- при удовлетворении какого условия следует выполнять тело цикла:
  for (; i<=5;) ,
- как нужно изменять переменную-счётчик после каждого повторения тела цикла:
  for (; ; i=i+1) .

Записанный в заголовке цикла оператор присваивания вида i=i+1 стоит обсудить подробнее, так как не исключено, что он может показаться Вам странным. Переменная i здесь стоит и слева и справа от знака присваивания (знак «равно»). С толку может сбить именно знак «равно». Для человека, привычного к математическим формулам, эта запись, если воспринимать её именно как формулу, означает чушь! Левая и правая части этого «равенства» никогда не могут быть равны друг другу.

Это правда. Но здесь записана вовсе не формула, а оператор присваивания. И «расшифровывать» его надо так:

- прочитать из ячейки с именем i хранящееся там число;
- прибавить к нему единицу;
- результат сложения записать в ту же ячейку i (при записи нового числа прежнее число будет стёрто).

В результате, после выполнения оператора присваивания переменная i получит новое значение, на единицу больше того, которое имела до его выполнения.

Ничто не мешает программисту таким же образом увеличивать, или уменьшать значение переменной не только на единицу, но и на любое нужное число.

Кстати говоря, в языке C++ для увеличения значения какой-нибудь целой переменной на единицу есть специальная команда: «++». Так что, в приведённой выше программе можно было бы использовать и более краткую запись заголовка оператора цикла:

```
for (i=1; i<=5; i++)
```

Можете испытать эту возможность в программе, но поначалу не слишком увлекайтесь ею, чтобы не наделать ошибок.

2. Оператор цикла “while”.

Во многих случаях для организации цикла, вместо оператора “for”, бывает удобнее использовать оператор “while” (англ. while – пока). Этот оператор цикла записывается в виде:

```
while (<условие>)          //заголовок цикла
<оператор>;                //тело цикла
```

или, с составным оператором в «теле» цикла,

```
while (<условие>)
{
 <оператор1>;
```

```
    <оператор2>;
    . . .
  }
```

Работает оператор “while” так:

- проверяется указанное в скобках условие;
- если оно удовлетворяется, выполняется «тело» цикла - одиночный или составной оператор, и опять происходит переход к проверке условия;
- если условие не удовлетворяется – оператор цикла завершает свою работу и управление передается следующему за телом цикла оператору программы.

Кратко оператор цикла “while” можно описать словами: пока удовлетворяется условие, повторяй выполнение «тела» цикла.

Очень простой пример использования цикла while в ситуации, когда число оборотов цикла заранее задано, приведён ниже.

```
//Программа, выполняющая обратный отсчёт секунд
//перед пуском ракеты

#include <iostream>
using namespace std;
//подключение библиотеки Windows
#include <Windows.h>

int main()
{
   int k;
   k = 10;  //задание начальн. значения переменной цикла
   while (k >= 1)                    //заголовок цикла
         {
               cout << k << endl;     //тело цикла
               Sleep(1000);           //тело цикла
               k = k-1;               //тело цикла
         }
 //следующий за циклом оператор
   cout << "Pusk!!!" << endl;
}
```

Приведённая выше программа работает так:

1. переменной k целого типа, которая играет роль счетчика, присваивается начальное значение, равное 10;
2. пока k больше нуля, циклически выполняются три следующих действия:
   - значение переменной k выводится на экран;
   - работа программы приостанавливается на 1 секунду (1000 миллисекунд), затем продолжается;
   - переменной k присваивается новое значение, на единицу меньше предыдущего;
3. после завершения цикла на экран выводится команда «Пуск!»

В этой программе операторы тела цикла выполняются только при условии, что значение переменной k больше или равно единице. Поскольку на каждом «обороте» цикла происходит уменьшение значения k, рано или поздно это значение станет меньше единицы, условие перестанет удовлетворяться, и цикл завершится.

В теле цикла использован вызов специальной подпрограммы-функции Sleep() (англ. sleep - спи!). Эта функция приостанавливает работу программы на заданное число миллисекунд (это число указывается в скобках). Такая «задержка» применена, чтобы отсчёт секунд до

пуска ракеты происходил медленно. Если этого не сделать, компьютер произведёт отсчет от 10 до 1 со скоростью работы центрального процессора, то есть безумно быстро. Чтобы использовать библиотечную функцию Sleep(), необходимо подключить дополнительную библиотеку подпрограмм. Это сделано в строке

#include <Windows.h>

Однако, вернёмся к «секретам» самого оператора цикла. Когда работает оператор while, проверка условия, указанного в скобках, происходит перед каждым выполнением тела цикла. Если уже изначально указанное условие не удовлетворяется, тело цикла не будет выполнено ни разу. И наоборот, если условие удовлетворяется при любых обстоятельствах – цикл будет бесконечным. Бесконечный цикл, обычно, возникает вследствие ошибки программиста, и чаще всего, нежелателен, так как остановить его можно только с помощью кувалды.

Чтобы цикл имел окончание, в его теле должны выполняться действия, меняющие значение той переменной, которая участвует в условии цикла. Этой переменной нужно присвоить нужное значение еще до начала работы оператора цикла, иначе проверка условия будет невозможной или бессмысленной. Такое присваивание называют инициализацией переменной цикла (англ. initial – начальный). В приведённой выше программе, от начального значения переменной цикла зависит количество оборотов цикла. Задав другое начальное значение этой переменной, Вы вполне можете отложить старт ракеты, например, на несколько минут, часов или лет.

Поэкспериментируйте с программой. Попробуйте задать такое начальное значение переменной цикла, чтобы тело цикла выполнилось всего один раз. Затем попробуйте задать его так, чтобы тело цикла не выполнилось ни разу.

Разумеется, в серьёзных программах циклы используют вовсе не для того, чтобы просто сосчитать от 10 до 1 или что-то в этом роде. В теле цикла обычно выполняют много и других полезных действий, ради повторения которых оператор цикла и вводят в программу.

3. Как быстро может работать оператор цикла?

Давайте разработаем компьютерную модель пешехода, и предложим ему «сходить» до Луны и обратно.

Моделью в науке и технике принято называть более или менее точную копию явления или объекта, например, самолёта или экономики большой страны. С моделью можно играть, делать опыты и т. д. Она может иметь облик оригинала: что-то вроде куклы, а может быть и просто совокупностью формул и правил, отражающих поведение объекта (математическая модель).

Компьютерная модель пешехода, которую мы хотим создать, должна уметь циклически производить шаги, считать их, и кроме того, подсчитывать пройденное расстояние и потраченное время. При достижении пункта назначения модель пешехода должна уметь останавливаться и сообщать нам результаты подсчетов. Займемся конструированием такой компьютерной программы.

Для ее работы понадобятся несколько переменных:

- счетчик шагов (целого типа),
- переменная для хранения пути, пройденного на текущий момент (двойной точности),
- переменная для хранения времени, прошедшего после старта (двойной точности).

Общая структура будущей «пешеходной» программы такова:

описание переменных,
инициализация переменных,
циклическое производство шагов от старта до финиша,
вывод на экран окончательных результатов.

Самая «умная» часть этой программы – производство шагов. Ясно, что для их повторения (для «ходьбы») понадобится оператор цикла. Один оборот цикла – один шаг пешехода. Какой вид оператора цикла выбрать?

Это зависит от того, знаем ли мы, отправляя пешехода «на задание», сколько пешеходу предстоит сделать шагов, и, следовательно, сколько понадобится оборотов цикла. Очевидно, что заранее количество шагов до Луны и обратно нам неизвестно, и выяснится только тогда, когда пешеход уже вернётся. Отсюда следует, что оператор цикла “for” для нашей программы не очень подходит. Обычно он используется тогда, когда число оборотов цикла задано. Похоже, что в такой ситуации лучше использовать оператор цикла “while”.

Теперь нужно подумать о том, при выполнении какого условия цикл должен завершаться? Ведь пешеход когда-нибудь останавливается.

Если поразмышлять отвлечённо, то постановка задачи о движении пешехода может быть двоякой:

а) можно предложить пешеходу идти в течение заданного времени, чтобы узнать, в итоге, какой путь ему удалось при этом пройти;
б) можно предложить пешеходу пройти заданный путь, чтобы узнать, в итоге, какое время он на это потратил.

Тогда, математическое условие продолжения (а не завершения) работы цикла, соответственно, может иметь двоякий вид:

а) текущее время - меньше, чем заданное время; например, $t < 60$;
б) пройденный к текущему времени путь - меньше заданного; например, $s < 100$.

Как только условие продолжения работы цикла перестаёт удовлетворяться – цикл завершается.

Тело цикла, очевидно, должно быть составным. В нём должны стоять операторы, выполняющие действия по подсчёту количества сделанных шагов, пройденного пути, прошедшего времени.

Чтобы определить итоговые затраты времени в годах, если поход весьма дальний, нужно решить, может ли пешеход идти круглые сутки.

В принципе, на ходу можно делать все насущные дела, кроме одного: нельзя на ходу спать. Ну, хотя бы потому, что во сне легко сбиться с дороги. Итак, из 24 часов нужно вычесть 8 часов на сон, тогда на движение остаётся 16 часов в сутки. Будем исходить из того, что час – 3600 секунд, а год – 365 суток.

Ниже приведена программа, моделирующая движение пешехода. В ней использованы следующие характеристики пешехода-человека: длина шага - 0,7 метра, время одного шага - 0,5 секунды. В качестве длины пути, который должен проделать пешеход для небольшой пробы сил, задано расстояние, соответствующее длине земного экватора: 40000 км, то есть 40000000 м.

Итак, для пробы, пешеходу предстоит совершить кругосветное путешествие. Только не надо уверять, что по океанам пешком не ходят: это у нас всего лишь игра.

//Компьютерная модель пешехода

```
#include <iostream>
using namespace std;
int main()
{
 //Описание переменных
        int  i;
        double  s,t;

 //Инициализация переменных
        i = 0;              //счётчик шагов
        s = 0;              //пройденный путь, в метрах
        t = 0;              //прошедшее время, в секундах
//Сообщение о начале движения
       cout << "Start!" << endl;
 //Циклическое производство шагов
        while ( s < 40000000 )      //условие движения
        {
            i = i + 1;      //подсчёт еще одного шага
            s = s + 0.7;  //увеличение пройденного пути
           t = t + 0.5;    //увеличение прошедшего времени
        }
//Сообщение об окончании движения
        cout << "Finish!" << endl;
//Вывод итоговых результатов
        cout << "Kolichestvo shagov     " << i;
        cout << endl;
        cout << "Projdennoe rasstojanie, km      " << s/1000;
        cout << endl;
        cout << "Potrachennoe vremia, let        ";
        cout << t/(3600*16*365);
        cout << endl;
}
```

Приведённая выше программа – это типичная программа-сумматор. Она циклически суммирует количество шагов, расстояние, время и выдаёт итог, аналогично тому, как кассовый аппарат в магазине суммирует цены покупаемых товаров и выдаёт их общую стоимость. Разница лишь в том, что суммируемые цены, обычно, отличаются одна от другой, а суммируемые нашей программой шаги – все одинаковой длины.

Итак, запустите программу. Она должна «прошагать» вокруг земного шара и сообщить Вам, что сделано примерно 57 миллионов шагов и на это потрачено чуть больше одного года. Если у Вас так и получилось, значит, программа работает правильно.

Теперь обратимся к первоначальной задаче. Мы хотим, чтобы пешеход сходил до Луны и обратно по волшебной дороге, по лунному лучу. Какое расстояние должны мы задать в программе?

Как известно, Луна движется вокруг Земли не по окружности, а по эллипсу, поэтому расстояние между Землёй и Луной не остается постоянным. Минимальное расстояние от Земли до Луны – 356400 км, максимальное – 406700 км. Примем среднее расстояние в 380 тысяч километров. Тогда путь до Луны и обратно будет составлять 760 тысяч километров.

Замените в тексте программы расстояние 40 тысяч километров на 760 тысяч километров (не ошибитесь с количеством нулей: ведь расстояние нужно указать в метрах), и запустите программу. Возможно, Вам интересно узнать, насколько быстро выполняет действия Ваш компьютер. Ему предстоит проделать огромную работу: выполнить примерно миллиард «шагов» - миллиард оборотов цикла. Замерьте по часам с секундной

стрелкой, за какое время он управился. В итоговом выводе на экран посмотрите, сколько лет потратил бы пешеход-человек на путешествие до Луны и обратно.

Задания к лабораторной работе:

1. **Откомпилировать и запустить все программы, приведённые в описании работы**

   В отчете представить:
   - исходные тексты программ (выделить цветными карандашами разным цветом заголовок цикла, тело цикла);
   - результаты работы программ - как они были выведены в окно консоли;
   - результаты замеров времени выполнения программы-пешехода и определение количества оборотов цикла в секунду.

2. **Составить программу, вычисляющую и выводящую на экран квадраты всех натуральных чисел от 1 до 20.**

   В отчете представить текст программы и результаты вывода на экран.

ЗАДАЧИ ПО ПРОГРАММИРОВАНИЮ

В этом задании требуется, соответственно своему номеру варианта, самостоятельно составить три программы:

1) с простыми вычислениями;
2) с условным оператором:
3) с оператором цикла.

Входные данные, если они в задаче есть, должны вводиться пользователем с клавиатуры. Программа должна производить необходимые действия и выводить результат на экран. В отчете должны быть приведены: текст задания, алгоритм решения задачи, исходный текст программы, результаты контрольных запусков программы. Защита отчета производится в компьютерном классе с демонстрацией работы программы.

1. Задачи для составления программы с простыми вычислениями

1. Полный бидон с молоком весит p кг. Бидон, наполненный наполовину, весит q кг. Сколько весит пустой бидон?
2. Во сколько раз n часов больше, чем m минут?
3. Черепаха Тортилла поселилась на n-ом этаже нового дома. Когда она идет гулять, то с этажа на этаж спускается за 0,5 часа. Сколько времени тратит бедное животное, чтобы выйти вниз на прогулку?
4. В двух карманах денег поровну. Из левого кармана в правый переложили k рублей. На сколько рублей в правом кармане стало больше, чем в левом?
5. У бумажного треугольника отрезали n углов. Сколько углов осталось?

6. Сколько разрезов потребуется, чтобы разделить на k частей бублик?
7. Периметр квадрата равен p метров. Какова его площадь в квадратных сантиметрах?
8. На озере росли лилии. Каждый день их число удваивалось, и на n -ый день заросло все озеро. На какой день заросла четверть озера?
9. Врач прописал больному k уколов через каждые полчаса. Первый укол сделали в m часов. Когда сделают последний укол?
10. Даны два действительных числа a и b. Получить их сумму, разность и произведение.
11. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
12. Даны два действительных положительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.
13. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.
14. Определить время падения камня на поверхность земли с высоты h.
15. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.
16. Вычислить период колебаний маятника длины L.
17. Даны гипотенуза и катет прямоугольного треугольника. Найти второй катет и радиус вписанной окружности.
18. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
19. Сколько лет исполнится сестре, когда брату будет n лет, если сейчас ему m лет, а сестра на три года моложе?
20. Сколько получится десятков, если n десятков умножить на m десятков?

2. Задачи для составления программы с условным оператором

1. Даны два символа (латинские буквы). Вывести их на экран в порядке возрастания по алфавиту.
2. Даны действительные числа x, y. Вывести на экран то из них, которое меньше.
3. Дано действительное число y. Вывести на экран его абсолютное значение.
4. Даны три символа (латинские буквы). Вывести на экран старший из них по алфавиту.
5. Даны действительные числа x, y, z. Вывести их на экран наименьшее из них.
6. Даны действительные числа a, b, c. Проверить, выполняются ли неравенства a<b<c.
7. Даны действительные числа a, b, c. Удвоить эти числа, если a> b > c, и заменить их абсолютными значениями, если это не так.
8. Даны два действительных числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.
9. Даны два действительных числа. Заменить первое число нулем, если оно меньше или равно второму, и оставить числа без изменения в противном случае.
10. Даны три действительных числа. Вывести на экран те из них, которые принадлежат интервалу (1, 3).
11. Даны действительные числа x, y. Меньшее из этих двух чисел заменить их полусуммой, а большее – их удвоенным произведением.

12. Даны три действительных числа. Возвести в квадрат те из них, значения которых неотрицательны.
13. Даны действительные числа x, y. Если они оба положительны, вывести их на экран. В противном случае вывести на экран их произведение.
14. Даны действительные числа x, y. Вывести их на экран в порядке убывания.
15. Даны действительные числа x, y. Если они имеют разный знак, вывести их на экран. В противном случае вывести на экран их произведение.
16. Даны координаты точки на плоскости x,y. Определить, сверху или снизу от оси x находится эта точка.
17. Даны площадь круга и площадь квадрата. Определить, поместится ли этот круг внутри этого квадрата.
18. Даны гипотенуза и катет прямоугольного треугольника. Определить, может ли такой треугольник быть построен.
19. Даны радиус окружности и сторона равностороннего треугольника. Определить, можно ли такой треугольник уместить внутри такой окружности.
20. Даны два символа (латинские буквы). Вывести на экран сообщение, упорядочены ли они по алфавиту.

3. Задачи для составления программы с оператором цикла

1. Дано натуральное число n. Вычислить: 2^n.
2. Улитка взбирается на дерево высотой h метров. В течение дня она поднимается на 5 м, но за каждую ночь опускается вниз на 4м. На какой день улитка достигнет вершины дерева?
3. Даны действительное число a, натуральное число n. Вычислить: a(a+1)…(a+n-1).
4. Вывести на экран заданное количество пронумерованных «приветов».
5. Даны действительное положительное число a, натуральное число b. Определить, сколько раз число b можно вычесть из числа a, чтобы остаток был не меньше нуля.
6. Расстояние между двумя телеграфными столбами равно x метров. Сколько телеграфных столбов нужно установить на промежутке в L метров?
7. Вычислить косинус заданного угла с помощью суммирования ряда.
8. Вывести на экран таблицу синусов угла от 1 до n градусов с интервалом 1 градус.
9. Вывести на экран таблицу корней квадратных из всех натуральных чисел от n до m.
10. Методом перебора найти наименьшее натуральное число, куб которого больше, чем число 9876543210.
11. Найти сумму синусов всех следующих углов : 10 20 30 40 50 60 70 80 90. (Углы указаны в градусах).
12. Увидит Миша где-нибудь брошенного котенка, непременно подберет и принесет к себе. Всегда воспитывается у него несколько котят; он не любил говорить товарищам — сколько, чтобы над ним не смеялись. Бывало, спросят у него:
    — Сколько у тебя теперь всех котят?
    — Немного,— ответит он,— три четверти их числа да еще три четверти одного котенка, вот и всего котят у меня. Товарищи думали, что он просто

балагурит. А между тем Миша задавал им задачу, которую нетрудно решить. Попытайтесь!

13. Даны действительное число a, натуральное число n. Вычислить: a^n.
14. У меня сестер и братьев поровну. А у моей сестры вдвое меньше сестер, чем братьев. Сколько нас?
15. — Дай мне яблоко, и у меня будет вдвое больше, чем у тебя,— сказал один школьник другому.
    — Это несправедливо. Лучше дай ты мне яблоко, тогда у нас будет поровну,— ответил его товарищ.
    Можете ли вы сказать, сколько у каждого школьника было яблок?
16. Сложить все натуральные числа от 1 до n.
17. Пионер собрал в коробку пауков и жуков — всего восемь штук. Если пересчитать, сколько всех ног в коробке, то окажется 54 ноги. Сколько же в коробке пауков и сколько жуков?
18. Дано натуральное число n. Вычислить: n!.
19. Теперь мой сын моложе меня втрое. Но пять лет назад он был моложе меня в четыре раза. Сколько ему лет?
20. Методом перебора найти наименьшее натуральное число, корень из которого больше, чем число 76,54321.

ПРИЛОЖЕНИЕ

Среда программирования Visual C++

В Visual C++ всю работу по составлению программы нужно выполнять в проекте. Проект – это своего рода коробка-контейнер, в которую будет помещена программа. Первый шаг в написании программы состоит в создании нового проекта и выборе типа проекта. Это важно, поскольку для каждого типа проекта Visual C++ устанавливает свои особые настройки компилятора (программы, переводящей с языка C++ на машинный язык).

Создание нового проекта

1. В меню **File (Файл)** выберите **New (Новый)**, и затем кликните мышкой на **Project (Проект).**
2. В области **Project Types (Типы проекта)** кликните **Win32**. Затем в панели **Templates** кликните **Win32 Console Application (Консольное приложение)**.
3. В строке **Name (Имя)** наберите латинскими буквами имя проекта. Например, **pr1**.

   Когда вы создаете новый проект, Visual C++ помещает проект в еще один контейнер под названием «решение» (**Solution**). В строке **Solution Name (Имя решения)** примите для решения имя по умолчанию, которое совпадает с именем проекта. В строке **Location (Размещение)** Вы можете выбрать папку, в которой хотите сохранить проект, или принять папку, предложенную по умолчанию.

Нажмите **OK** , чтобы запустить **Win32 Application Wizard (Волшебник создания приложения для Win32)**.

4. На странице **Overview** в диалоговом окне **Win32 Application Wizard** кликните **Next (Дальше)**.
5. На странице **Application Settings** под заголовком **Application type (Тип приложения)**, выберите **Console Application (Консольное приложение)**. Выберите установку **Empty Project (Пустой проект)** под заголовком **Additional options (Добавочные свойства)** и кликните **Finish (Завершить)**.

Сейчас Вы создали проект, пока ещё не содержащий файл с исходным текстом программы, - пустой проект. Второй шаг состоит в подготовке исходного текста. Это можно сделать с помощью встроенного в Visual C++ текстового редактора.

Создание файла с исходным текстом программы и добавление его к проекту

1. В меню **Project** кликните пункт **Add New Item (Добавить новый элемент)**.

   В поле **Categories: Visual C++** выберите **Code (Программа)**. Затем кликните на **C++ File (.cpp)**.
2. Наберите имя нового файла в поле Name, например, **p1**, и кликните **Add (Добавить)**.
3. В появившемся окне редактора текстов наберите исходный текст своей программы на C++.

На следующем шаге нужно выполнить перевод программы на машинный язык (компиляцию) и объединение ее с библиотечными подпрограммами (компоновку).

Компиляция и компоновка программы

1. В меню **Build (Построить)**, кликните **Build Solution (Построить Решение)**.

   После этого в окне **Output (Вывод)** начнет отображаться информация о ходе компиляции и компоновки, информация о расположении протокола компиляции и компоновки, сообщения компилятора и компоновщика о результатах их работы.
2. Убедитесь, что в окне **Output** появилось сообщение: pr1 - 0 error(s), 0 warning(s) (0 ошибок, 0 предупреждений). Это сообщение говорит о том, что компиляция программы и ее компоновка прошли успешно. Если в сообщении указано ненулевое количество ошибок, значит в тексте Вашей первой программы Вы где-то допустили опечатку. Найдите опечатку, исправьте ее и повторите компиляцию и компоновку.

После этого шага Ваша первая машинная программа полностью готова. Файл с программой **pr1.exe** находится во вложенной в папку проекта папке Debug, куда его поместил компоновщик. Можно «вручную» сразу запустить этот файл на исполнение, но лучше воспользоваться специальными средствами запуска, встроенными в Visual C++.

Запуск программы

В меню **Debug (Отладить)**, кликните **Start without Debugging (Запустить без отладки)**.

При этом на рабочем столе Windows откроется окно консоли, где белыми буквами на черном фоне будут выведены результаты работы Вашей первой программы. Окно консоли закроется только после того, как будет нажата какая-либо клавиша. Если запускать программу другим способом, окно консоли закроется немедленно после вывода результата, и Вы не успеете его рассмотреть.

Если это необходимо, можно снова вернуться в окно текстового редактора, изменить нужным образом исходный текст программы и опять повторить все этапы.

Сохранение результатов работы

По окончании работы, в меню **File** кликните пункт **Save All (Сохранить всё)**.

При этом будут сохранены все файлы проекта, включая файл с исходным текстом программы и файл с окончательной программой на машинном языке.

В какой именно папке будет сохранен проект, зависит от заданных Вами настроек. Если Вы не настраивали путь сохранения и не знаете, где после сохранения оказались файлы проекта, то в любом случае сможете найти их с помощью имеющегося в главном меню Windows средства «Поиск».

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Информатика. Базовый курс./ Под ред. С.В. Симоновича. – СПб.: Питер, 2005.

Брайан Оверленд. C++ без страха. – М.: Изд-во Триумф, 2005.

Учебное издание

Сивков Анатолий Михайлович

ПРОГРАММИРОВАНИЕ.
РУКОВОДСТВО ПО ВЫПОЛНЕНИЮ
ЛАБОРАТОРНЫХ РАБОТ

Авторская редакция

Пописано в печать 00.00.00. Формат 60 × 84 $^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 0,00. Уч.-изд. л. 0,00.
Тираж 00 экз. Заказ № 0000.

Издательство «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, д. 1, корп. 4.